Updated 11/7/2020

MET CS682 TERM PROJECT PART 2

Design with UML

Sneha

The purpose of this exercise is to give you practice modeling with UML tools. The assignment builds on last week’s scenario, utilizing a system you are familiar with so that you can concentrate on UML modeling.

You are to specify UML design for *RemoteAssist* that assists employees in their remote work.

The following characteristics apply:

* Your scope from term project part 1. *RemoteAssist* has a potentially large scope but your solution should focus on an aspect or aspects of your choice.
* Your solution should focus on software-intensive aspects.

The last section below contains numerous hints.

# Use Cases, Revised

## Improve and update the two use cases from Term Project Part 1 if you can (or replace them if you prefer). These will be used for the rest of the assignment.

## First Use Case

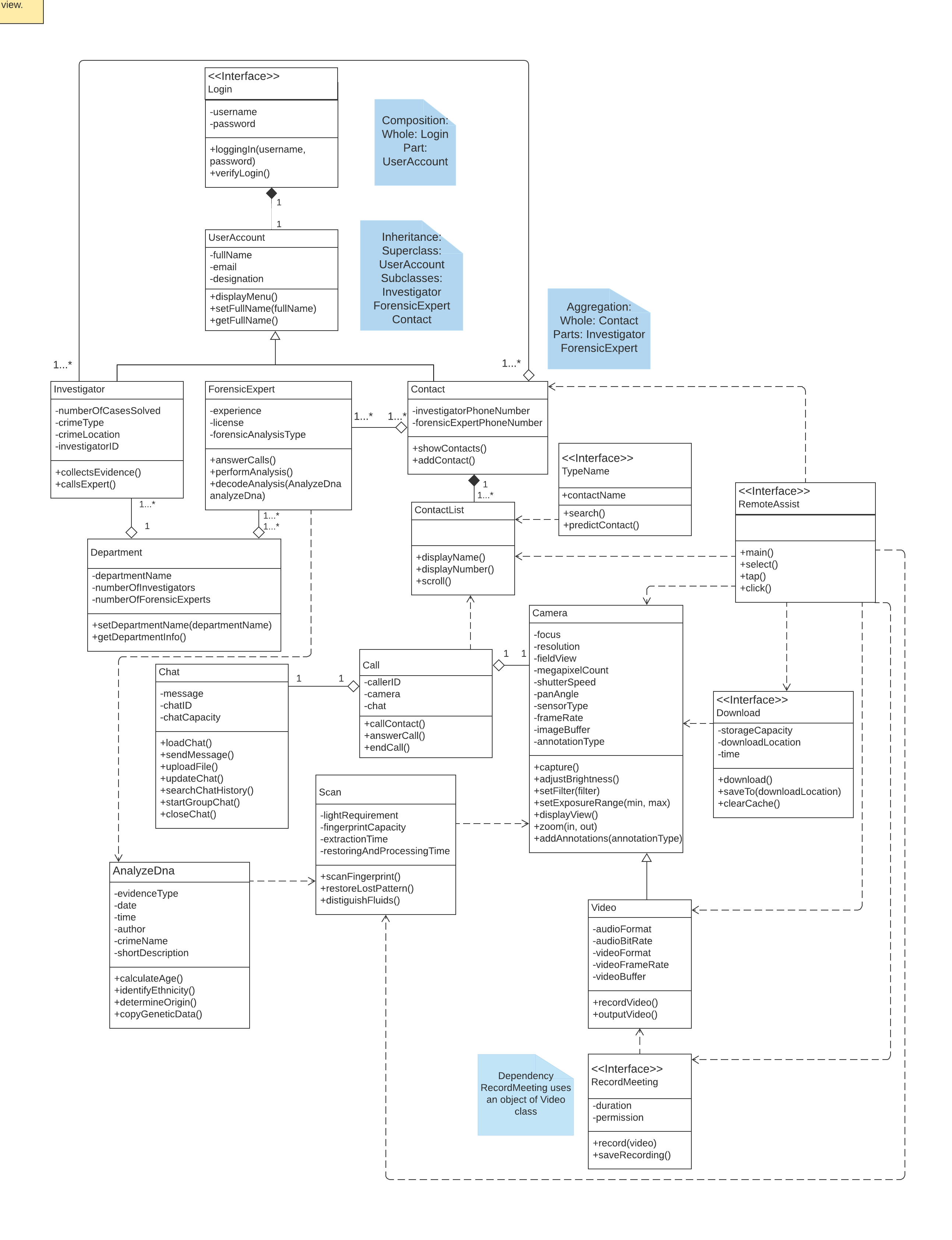
|  |  |  |
| --- | --- | --- |
| **Use case Name** | Investigator (Caller) Use Case | |
| **Actor:** | Investigator | |
| **Description:** | RemoteAssist shall allow the investigator to contact a user in another department. | |
| **Pre-condition:** | The investigator logs into the application on a smart display device. RemoteAssist shall open the home screen displaying the menu. | |
| **Step #** | **Actor** | **System** |
| **1** | The investigator clicks on Contacts. | *RemoteAssist* displays a contact list. |
| **2** | The investigator starts typing the contact’s name in the search bar. | *RemoteAssist* starts searching the name and predicts the name, simultaneously. |
| **3** |  | *RemoteAssist* returns a list of contacts matching the name entered. |
| **4** | The investigator selects a contact and places a call. | *RemoteAssist* calls the selected contact. |
| **5** |  | *RemoteAssist* returns the call response. In this case, the recipient answers the call. |
| **6** | The investigator clicks on the camera icon. | *RemoteAssist* opens the camera and displays field view. |
| **7** | The investigator clicks on the icon that scans fingerprint. | *RemoteAssist* scans and processes the fingerprint and restores any lost patterns. |
| **8** | The investigator clicks on the video icon. | *RemoteAssist* switches to video mode. |
| **9** | The investigator clicks on the record button. | *RemoteAssist* records the video as a meeting. |
| **10** | The investigator speaks and clicks on the End Call button. | *RemoteAssist* ends the recording and disconnects from the call. |
| **Alternate Courses:** | #4 The investigator’s call is not answered— display message “Call Disconnected.”  #7 If *RemoteAssist* fails to restore lost patterns of a fingerprint, display “Unable to restore fingerprint patterns! Try again.” | |
| **Implementation Constraints:** | 1. The request for call must be managed within 3 seconds. 2. *RemoteAssist* shall have a high-resolution camera of 50 megapixels or higher. 3. *RemoteAssist* shall not take more than 10 seconds to process the fingerprint patterns. | |

## Second Use Case

|  |  |  |
| --- | --- | --- |
| **Use case Name** | Forensic Expert (Recipient) Use Case | |
| **Actor:** | Forensic Expert | |
| **Description:** | *RemoteAssist* shall allow forensic expert(s) to answer calls to solve problems in real-time, pull crucial evidence into view, and support collective problem-solving. | |
| **Pre-condition:** | The forensic expert will always be logged in to the application on a smart display device. The device (system) will display the menu on the home screen. | |
| **Step #** | **Actor** | **System** |
| **1** | The forensic expert receives a call. | *RemoteAssist* notifies the user by ringing. |
| **2** | The forensic expert answers a call. | *RemoteAssist* displays the call screen. |
| **3** |  | *RemoteAssist* displays a camera view showing the image of fingerprints captured by the investigator. |
| **4** |  | *RemoteAssist* displays the processed image of the fingerprints. |
| **5** | The forensic expert clicks on the download icon. | *RemoteAssist* makes a copy of the image by downloading it. |
| **6** | The forensic expert clicks on DNA test icon. | *RemoteAssist* starts analyzing fingerprints and returns DNA test results. |
| **7** | The forensic expert decodes the test results to the investigator by sending the details in the chat box. | *RemoteAssist* sends the message to the investigator and displays the message to both the users. |
| **8** |  | *RemoteAssist* notifies the expert the call is being recorded. |
| **9** | The forensic expert ends the call. | *RemoteAssist* ends the recording and disconnects from the call. |
| **Alternate Courses:** | #2 The forensic expert may not answer the call.  #5 If the DNA test does not return any data—display “Match not found” | |
| Implementation Constraints: | 1. The system shall not take more than 15 seconds to provide the DNA test results. 2. Messages shall not take more than 2 seconds to deliver. | |

# Class Model

Provide a class model for the *RemoteAssist* system, maintaining the system scope you determined. Your solution should have approximately 10-12 classes. (When complete, a real design typically contains perhaps hundreds of classes, so your submission will have to focus on the scope of your choice.) Your class model should show classes and their relationships, including at least one inheritance. To add clarity to your diagram, provide key attributes and methods—you do not need to list every attribute and or method, just the most important ones. Label everything appropriately and clearly.



# Documenting Classes and Relationships

Select three most important entity classes and one non-entity design class from the class model and the format below, describe the reason why it was important to select in your design and any important design choice decisions around relationships with other classes. Consider classes which are complex within your design and require additional explanation as part of your selection.

**Entity 1: UserAccount**

**Reason for selection**: The “UserAccount” class encourages inheritance-specialization relationship. The key reason is for the “UserAccount” class to contain attributes and methods that can be reused by its subclasses: “Investigator” class, “ForensicExpert” class, and “Contact” class. The “Investigator” class and the “ForensicExpert” class contains the main users of the system inheriting the attributes and methods from the “UserAccount” class.

Attributes:

* fullName: contains full name of the users.
* email: contains email address of the users.
* designation: contains the job title of the users.

Methods:

* displayMenu(): displays menu on the home screen.
* setFullName(fullName): establishes the given value as the full name of a user.
* getFullName(): retrieves the full name of a user.

**Relationship design choices with other classes**:

* The “UserAccount” bonds with the “Login” non-entity interface by sharing a composition (has-a) relationship. Any changes in one of the entities will affect both the entities.
* The “UserAccount” class supports the concept of inheritance-specialization, where it acts as a super class to its specializations or subclasses:“Investigator”, “ForensicExpert”, and “Contact.” The subclasses will inherit all the attributes and methods of the “UserAccount” class.

**Entity 2**: **Investigator**

**Reason for selection**: The “Investigator” class derives all the attributes and methods from its superclass “UserAccount” and has its own set of attributes and methods. The main reason for defining the “Investigator” class is to include attributes and methods that are relevant to investigators and their work.

Attributes:

* numberOfCasesSolved: shows the number of crimes resolved by an investigator.
* crimeType: covers the category of the crime that is currently under investigation.
* crimeLocation: holds the location of the occurrence of a crime.
* investigatorID: contains identification number of an investigator.

Methods:

* collectEvidence(): stores evidences collected at a crime scene listing the types of evidences found: fingerprints, hair strands, or blood and fluids.
* callExpert(): calls a forensic expert.

**Relationship design choices with other classes**:

* The “UserAccount” class is the parent class for the “Investigator” class. The “Investigator” class automatically derives all the attributes and methods from its parent class due to inheritance.
* The “Investigator” class shares an aggregation (has-a) relationship with the “Department” class. The “Department” class is thought of as a “whole” and the aggregated class “Investigator” is thought of as a “part.” The multiplicity reads each investigator belongs to only one department.
* The “Investigator” class shares an aggregation relationship with the “Contact” class. The “Contact” class is thought of as a “whole” and the aggregated class “Investigator” is thought of as a “part.” Each investigator may have one or more contact numbers. The aggregated object is *investigator* which is a part of the “Contact” class.

**Entity 3**: **ForensicExpert**

**Reason for selection**: The “ForensicExpert” class derives all the attributes and methods from its parent class “UserAccount” and has its own set of attributes and methods. The main reason for defining the “ForensicExpert” class is to include attributes and methods that are relevant to a forensics team.

Attributes:

* license: contains the license number of a forensic expert.
* experience: shows the level of expertise of a forensic expert.
* forensicAnalysisType: contains on the type of forensic investigation: hair and fiber analysis, fingerprint matching, or blood analysis.

Methods:

* answerCalls(): answers incoming calls from other departments.
* performAnalysis(): performs DNA analysis on the digital evidences.
* decodeAnalysis(): translates the DNA results in a comprehendible form.

**Relationship design choices with other classes**:

* The “UserAccount” class is the parent class for the “ForensicExpert” class. The “ForensicExpert” class automatically derives all the attributes and methods from its parent class due to inheritance.
* The “ForensicExpert” class shares an aggregation relationship with the “Department” class. The “Department” class is thought of as a “whole” and the aggregated class “ForensicExpert” is thought of as a “part.” The multiplicity reads each investigator belongs to one or more departments.
* The “ForensicExpert” class shares an aggregation relationship with the “Contact” class. The “Contact” class is thought of as a “whole” and the aggregated class “ForensicExpert” is thought of as a “part.” Each forensics expert may have one or more contact numbers. The aggregated object is *forensicsExpert* which is a part of the “Contact” class.
* The “ForensicExpert” class shares a dependency relationship with the “AnalyzeDna” class. The *decodeAnalysis*() method in the “ForensicExpert” class uses an object of the “AnalyzeDna” class.

**Entity 4**: **Contact**

**Reason for selection**: The “Contact” class derives all the attributes and methods from its superclass “UserAccount”. The reason for selecting the “Contact” class is to portray a vital functionality of the system, to connect investigators with forensic experts. The “Contact” class has a set of attributes and methods specific to the class.

Attributes:

* investigators: contains contact details of investigators.
* forensicExperts: contains contact details of forensic experts.

Methods:

* showContacts: loads all the contacts in the directory.
* addContacts: adds new contacts to the directory.

**Relationship design choices with other classes**:

* The “Contact” class is acts as a “whole” for the classes: “Investigator”, “ForensicExpert,” and “ContactList.”
* The “Contact” class shares an aggregation (has-a) relationship with its parts: the “Investigator” class and the “ForensicExpert” class. These parts can exist independent from the whole.
* The “Contact” class shares a composition relationship with the “ContactList” class. The part— “ContactList” class—cannot exist without its whole—the “Contact” class.

**Non-Entity 1: Download**

**Reason for selection**: The reason for selecting the non-entity interface “Download” is to download and save images through it.

Attributes:

* storageCapacity: allocates the maximum storage available for downloading images. The capacity is measured in bytes.
* time: allots the time to be taken for each download.
* downloadLocation: assigns the location to indicate where the downloads will appear.

Methods:

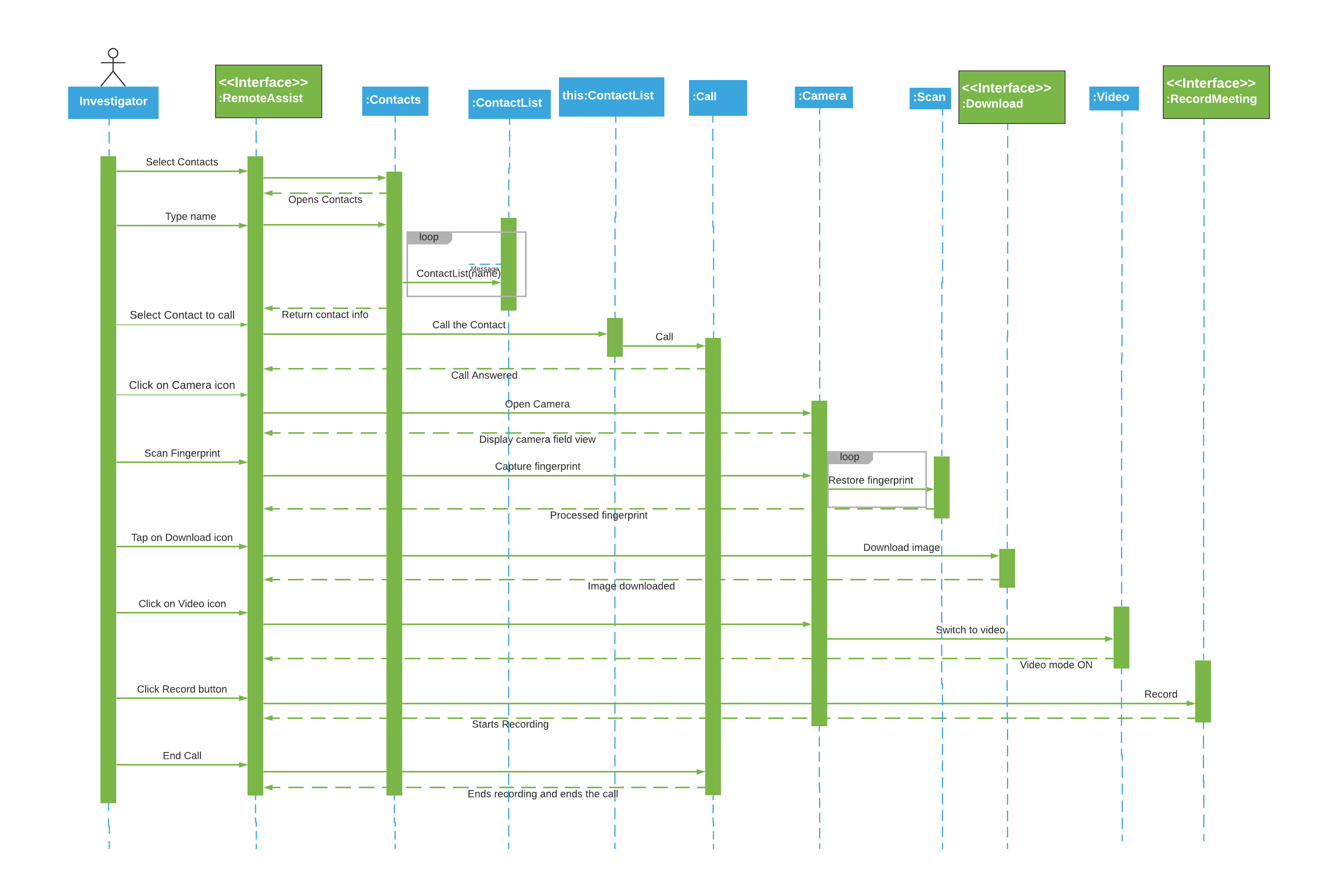
* download(): starts and completes downloading of images when the download icon is triggered.
* saveTo(downloadLocation): saves the downloads in the given location.
* clearCache(): clears data and cache.

**Relationship design choices with other classes**:

* The “Download” non-entity interface shares a dependency relationship with the “Camera” class. The “Camera” class provides methods to capture images after which the download can be initiated.

# Sequence Diagram

## Provide a detailed sequence diagram for one specific functional area of your choice. Your solution should have at least six objects, with function calls among them, and needs to be consistent with a use case in part one as well as classes identified in Section 4.1 (Sequence diagrams are frequently based on use cases.) The objects involved are instances of a classes that should be among those in the class model Section 4.2. Make sure to represent these correctly: for example, if you were to create an instance of an VideoChat class (i.e., to model a new video chat instance) it may be called “newVideo:Video” or just “:Video” if the instance name is relevant.



# Appendix

Since Term Project Part 2 builds on Term Project Part 1, Provide requirements you have outlined last week in Term Project Part 1. Not everything needs to be included, only the parts you feel are helpful in supporting the UML design. This section is not graded.

1. *RemoteAssist* shall allow users to login with their departmental credentials.
2. *RemoteAssist* shall allow the Investigator to contact other user(s) in real-time by chat, audio, or video.
3. *RemoteAssist* shall have a camera that will incorporate most basic settings of modern cameras.

* Panorama mode
* 3600 view
* Live Focus and Selective focus
* Slow Motion
* Hyper-lapse
* Self-timer and more

1. *RemoteAssist* shall have a camera feature with the following advanced image processing techniques.

* Capture and restore fingerprint patterns.
* Provide 3D images of shoe/ footprints.
* Describe hair strands accurately by indicating the color of the strand.
* Provide filters to set out unique wavelengths which projects substances from their environments. (The substances can be blood, water, alcohol, chemicals, etc.)

1. *RemoteAssist* shall allow the forensics expert to zoom in and out of the video during the call.
2. *RemoteAssist* shall allow both the investigator and the forensics expert to record a call.
3. *RemoteAssist* shall allow both the investigator and the forensics expert to send messages, images, documents, or videos in the chat.
4. *RemoteAssist* shall allow both the investigator and the forensics expert to download and screenshot images.
5. *RemoteAssist* shall save the downloaded images and videos and the call recordings in Files.

# References

Show that you used a wide variety of resources by listing them below and clearly indicating in the body above where you used. Make sure to use proper referencing in your paper. We suggest using APA format, but other formats are fine as long as it clearly distinguishes your work from work of others in your response—be mindful of plagiarism rules.

[1] Dzemydiene, Dale, et al. “Multiple Regression Analysis in Crime Pattern Warehouse for Decision Support.” ResearchGate, 15 Nov. 2020, www.researchgate.net/figure/Dynamics-of-the-main-social-economic-indicators-of-Lithuania-GDP-per-capita-is-presented\_fig3\_221464853.

[2] Dennis, A., Wixom, B. H., & Roth, R. M. (2015). Systems Analysis & Design. Wiley.

[3] Module 4: Object-Oriented Design and UML Modeling. (n.d.). Retrieved from <https://learn.bu.edu/bbcswebdav/pid-8604587-dt-content-rid-50582522_1/courses/21sprgmetcs682_o1/course/module04/allpages.htm>

[4] “UML Relationships with EXAMPLE: Dependency, Generalization, Realization.” Guru99, www.guru99.com/uml-relationships-with-example.html#7.

# Evaluation



**Please do not include Hints section in your solution.**

# Hints

## Overall Assignment Notes

* You may use Visio, LucidChart, or another design tool of your choice (please check with your facilitator in advance if the latter).
* In order to make your design clear to your reader, accompany each requested section with a brief description that explains your thinking and the choices that you made.
  + This is not required for the Use Case (section 1) but is required for all other sections.
* As usual, the notes are a primary source for explanations and examples; we also encourage you to do outside reading and research to gain additional perspective.
* Apply the usual editing process. It is tempting to repeat important points but this often weakens your work and feels to the reader like padding.
* Explain use of references in the appendix. Don't list sites that are not specifically referred to in the body of your paper.
* It is important that the entire solution is consistent. You will want to iteratively review all of the four sections to make sure that together they are consistent as a single solution.

## Use Cases

* Review Assignment 3 and the feedback that was provided

**Clarity**:

* Look to refine scope and clarify the steps (i.e. if someone else read it - would they understand this clearly?)

**Technical soundness:**

* The use case needs to show appropriate sequence (actor/system), branching needs to be minimal within alternate steps. Clear understanding in difference between constraints and pre-conditions. The use case itself should follow one path as best as possible.

**Relevance:**

* Are the use cases relevant and consistent to your class model and sequence diagram?

**Thoroughness and coverage:**

* Did you link to your requirements/user story (place in appendix) did you think through preconditions, alternate steps, and constraints.

## Class and Sequence diagrams

* Note that some students tend to identify classes by first writing sequence diagrams while others prefer to first identify class models, thus you are free to try different approaches.

**Relevance:**

* Are the classes relevant and consistent to your use case, class model, and sequence diagram? Is the scope of your design relevant to the solution? For example, choosing a use case, classes and methods that are secondary (for example, authentication).
* Focus on more entity classes vs. non-entity classes

**Technical Soundness:**

* A good place to start in identifying classes is to review Module 4 “Introduction to UML.” You will need to understand how to identify business objects and their relationships. Note how examples in the module start with the use case; this is where going back to the use case you developed is important. Take a use case and break it down into verbs and nouns.
* Solution needs to show that you understand the difference between entity and non-entity classes and that you have identified and organized these correctly. (an example is <<design>> for a GUI-related class).
* Review checklist for Sequence and Class diagrams—these will identify areas where many miss parts on technical soundness, clarity, and possibly thoroughness and coverage

**Clarity:**

* Explain your diagrams, especially parts that are not self-evident. For example, why is a specific relationship aggregation or composition? A good way to do this is to provide a short-bulleted list under the diagram itself.
* Are diagrams clear to read (i.e. no overlapping lines, no non-polished designs). In sequence diagrams, return messages should only be added as needed, for clarity.
* Make sure that you have a consistent design throughout the solution. Are all the classes and messages in the sequence diagram represented in the class diagram? Are these explained in your classes overview. Methods in the sequence diagram should be in the class diagram? All the class names should be consistent.
* Make sure that your sequence diagram is consistent between the use case and the classes which you have identified in previous sections.
* Try to balance clarity and thoroughness, as an example in sequence diagrams return messages may clutter your design.

**Thoroughness and coverage:**

* Flush out your diagrams with details where you can. For example, perhaps in a class Search, we have methods for detectingSearchCriteria() and performSearch().

## Specific to Class Model

**Technical Soundness:**

* Please see Module 4 primary readings as the textbook provides much useful detail around object-oriented design and class modeling- You will want to understand concepts such as—but not limited to—inheritance, generalization, specialization, aggregation, composition, and association.
* Show understanding between different types of relationships, for example, aggregation vs. inheritance. In situations where it is a design choice (i.e. aggregation vs. composition)
* Consider the use of dependency as a relationship between entity and non-entity classes. Dependency between entity classes should be explained.
* Multiplicity should be shown on aggregation and composition and no other type of relationship.

**Thoroughness and coverage:**

* It is helpful to add attributes and methods to your class diagram where appropriate, it will help you think through the classes and create more thoroughness and coverage within your solution.

## Specific to Sequence Diagram

**Technical Soundness:**

* Start by reading the Module 4 “Sequence Diagrams” section, as it has several examples and approaches. Module 6 has additional examples of detailed sequence diagrams if you need more examples.
* The textbook goes into some detail on sequence diagrams on pages 202-210.
* The function/method belongs in the class that the arrow points to—this should be represented in the class diagram.
* Functions (solid line, method name) and return messages (dashed line, description) correctly labeled. These should only be used for clarification.
* Objects involved in a sequence diagram are all instances of classes. In some situations (i.e., if in a sequence diagram there are multiple instances of the same class shown) it may be appropriate to label each individual instance of the class (e.g., NewCart:Cart and OpenCart:Cart)
* Classes correctly labeled—including stereotypes where appropriate (CamelCase, vs. camelCase, no spaces, etc)